Design of an Incremental Clustering Package for Protein Function and Family Analysis

Chien-Yu Chen
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

Hsueh-Fen Juan
Department of Chemical Engineering, National Taipei University of Technology, Taipei, Taiwan

Po-Jen Hsiao
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan,

Shui-Tein Chen
Institute of Biological Chemistry, Academia Sinica, Taipei, Taiwan

Hsiang-Wen Tseng
Institute of Biological Chemistry, Academia Sinica, Taipei, Taiwan
Institute of Pharmacology, National Yang-Ming University, Taipei, Taiwan

Yen-Jen Oyang
Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan

cychen@mars.csie.ntu.edu.tw and yukijuan@gate.sinica.edu.tw

Abstract

Proteins clustering has been widely exploited to facilitate in-depth analysis of protein functions and families. This paper discusses the design of an incremental protein clustering package that provides comprehensive features for protein function and family analysis. Specifically, the package offers alternative options for carrying out high-quality protein clustering from different aspects. The incremental nature of the clustering algorithm is essential for efficient analysis of those contemporary protein databases whose sizes are growing rapidly. Concerning the quality of clustering results, experimental results from applying the incremental clustering algorithm to protein sequence analysis show that the incremental algorithm is able to identify protein sequence clusters that match protein families more consistently than the single-link algorithm, which is the most widely used hierarchical clustering algorithm for protein sequence analysis. This paper also addresses the implementation techniques employed to improve the system performance.

Keywords
Protein clustering, clustering utility.

1. Introduction

The human genome project has opened novel scientific avenues such as proteomics. The major challenge in proteomics is to predict protein functions and structures. Individual functions assigned to different sequence segments (motif) combine to create a complex function for the whole protein [11]. "Motif" in protein chemistry was first coined in 1980 [10]. Using bioinformatics tool to search protein sequence motif was stared from 1995 [5].

With the scale of influx of raw sequence data from genome sequencing projects, manual annotation of all gene products (proteins) is no longer possible, and therefore there is a need for reliable, automatic methods for protein sequence analysis and clustering. While on-line resources are available for sequence searching, there are very few available for searching the small functional sites/motifs which are related with the functions of multiple-domain proteins. InterPro (integrated resource of protein domains and functional sites) is the integrated resource for protein secondary annotation such as protein families, domains and functional sites [2].

In this paper, we discuss the design of an incremental protein clustering package that provides comprehensive features for protein function and family analysis. Specifically, the package provides the function of searching proteins with the specified small functional sites/motifs and offers alternative options for carrying out high-quality protein clustering on these proteins from different aspects. The alternative options include different bases of similarity measurements and different agglomerative hierarchical clustering algorithms to invoke. With this package, biochemists can conduct extensive analysis of protein functions and families.

From algorithm design point of view, the incremental nature of the clustering algorithm is essential for efficient analysis of those contemporary protein databases whose sizes are growing rapidly. Concerning the quality of clustering results, the experimental results reported in this paper show that the incremental algorithm is able to identify protein sequence clusters that match protein families more consistently than the single-link algorithm, which is the most widely used traditional hierarchical clustering algorithm for protein sequence analysis [9]. This paper also addresses the two critical implementation techniques, shared memory and caching, employed to improve the system performance.

2. Similarity measure

2.1 Pairwise similarity of proteins

The clustering analysis is conducted based on two alternative similarity measures. The first similarity measure con-

Proceedings of the IEEE Fifth International Symposium on Multimedia Software Engineering (ISMSE'03)
0-7695-2031-6/03 $17.00 © 2003 **IEEE**

cerns the amino acid sequences of the proteins and the second similarity measure concerns the annotations associated with the proteins in the protein database. The similarity between two amino acid sequences is determined by invoking the BLAST utility [1]. On the other hand, if the user intends to cluster the proteins based on the annotations, then a feature vector is generated for each protein. The dimensions of the feature vector space correspond to the set of words present in the "key word (KW)" and "free text comments(CC)" fields of the Swiss-Prot database [4]. For a protein ***P***, its feature vector $<v_1, v_2, \ldots, v_d>$ is defined as follows:

$$v_i = f_i \times \log_2 \frac{|S|}{c_i} \quad (1)$$

, where f_i is the number of instances of word $\mathbf{w_i}$ in protein ***P***, S is set of proteins that contain the interested motif, and c_i is the number of proteins in S whose annotations contain word $\mathbf{w_i}$. In fact, equation (1) above is the common (*term-frequence*) × (*inverse-document-frequency*) term employed in information retrieval [3]. Once the feature vector of each protein is determined, we can compute the similarity between a pair of proteins accordingly. There are two common approaches in this regard as presented in the following:

(1) dissimilarity based on Euclidean distance,

$$dissimilarity(\hat{P}, \tilde{P}) = \sqrt{\sum_{i=1}^{d} (\hat{v}_i - \tilde{v}_i)^2},$$

(2) similarity based on the cosine value of the two feature vectors,

$$similarity(\hat{P}, \tilde{P}) = \frac{\sum_{i=1}^{d} \hat{v}_i \cdot \tilde{v}_i}{\sqrt{\sum_{i=1}^{d} \hat{v}_i^2} \cdot \sqrt{\sum_{i=1}^{d} \tilde{v}_i^2}}$$

, where $< \hat{v}_1, \hat{v}_2, ..., \hat{v}_d >$ and $< \tilde{v}_1, \tilde{v}_2, ..., \tilde{v}_d >$ are the feature vectors of proteins $\hat{P}$ and $\tilde{P}$, respectively.

Once the pair-wise similarities or dissimilarities between proteins have been computed, the conventional hierarchical agglomerative clustering algorithms are invoked to build a dendrogram.

2.2 Similarity measure between clusters

The incremental clustering algorithm employed in the utility will invoke a hierarchical agglomerative clustering (HAC) algorithm for building hierarchy. This utility provides three alternative clustering algorithms, namely, the single-link, the complete-link, and the average-link [7]. One of the main distinctions of this utility is that a summarization process will be conducted on the dendrograms generated by the clustering algorithms in order to make the final dendrogram easier to interpret. Most hierarchical agglomerative clustering algorithms share the same major procedures as follows [7]:

1. Compute the proximity matrix containing the distance between each pair of patterns. Treat each pattern as a cluster.
2. Find the most similar pair of clusters using the proximity matrix. Merge these two clusters into one cluster. Update the proximity matrix to reflect this merge operation.
3. If all patterns are in one cluster, stop. Otherwise, go to step 2.

Three widely used measures for similarity between clusters are as follows, where C_i or C_j stands for a particular cluster.

- Single-link: $dis(C_i, C_j) = \min_{p \in C_i, q \in C_j} dis(p, q)$
- Complete-link: $dis(C_i, C_j) = \max_{p \in C_i, q \in C_j} dis(p, q)$
- Average-link: $dis(C_i, C_j) = \frac{1}{|C_i| \times |C_j|} \sum_{p \in C_i} \sum_{q \in C_j} dis(p, q)$

3. Incremental hierarchical clustering

In the clustering utility, we employ an incremental hierarchical clustering algorithm proposed by [6]. Fig. 1 shows the major operations carried out by the incremental clustering algorithm presented in this paper. The algorithm consists of two phases of operations, namely, the initialization phase and the incremental phase. In the initialization phase, a set of instances extracted from the database is taken to construct the initial dendrogram. The number of data instances extracted could range from a few hundreds to a few thousands. With the initial set of objects, a conventional agglomerative hierarchical clustering algorithm, such as single-link or complete-link [7], is invoked to construct the initial dendrogram.

With the initial dendrogram, a summarization process is then conducted to identify a set of representatives that collectively provide an abstract description of the distribution of the samples. With these representatives, an abstract dendrogram is generated. The abstract dendrogram provides the basis for the incremental phase of the clustering algorithm to proceed. In the incremental phase, all the remaining data instances in the database, i.e. those objects that were not taken as samples, as well as the new objects that are continuously added into the database are examined one by one and the abstract dendrogram is updated dynamically to reflect the evolution of the database.

3.1 The summarization process

In the summarization process, clusters that meet the following criterion are identified as homogeneous clusters. Let random variable X_C corresponds to the observed pairwise similarity between two objects randomly selected from a cluster C. Then, the outcome space S_C of X_C contains $|C| \times (|C|-1)/2$ possible outcomes. $|C|$ stands for the number of objects in C. In statistics [8], the *skewness* and *kurtosis* of X_C, denoted by $Skew(X_C)$ and $Kurt(X_C)$ respectively, are

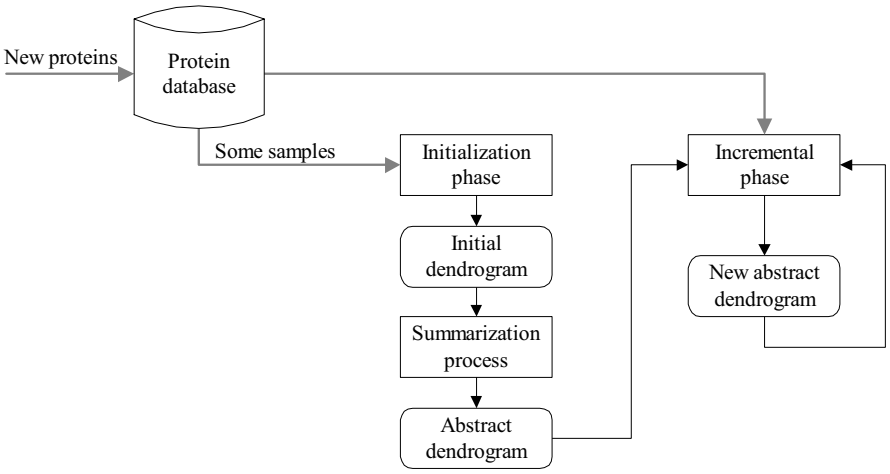


Fig. 1. A system diagram that summarizes the operations carried out by the proposed incremental clustering algorithm.

defined as follow:

$$Skew(X_C) = \frac{|C|}{(|C|-1)\times(|C|-2)} \sum_{x_i \in S_C} (x_i - \bar{x})^3 / s^3$$

$$Kurt(X_C) = \left[\frac{|C|(|C|+1)}{(|C|-1)(|C|-2)(|C|-3)} \sum_{x_i \in S_C} (x_i - \bar{x})^4 / s^4 \right] - 3\frac{(|C|-1)^2}{(|C|-2)(|C|-3)}$$

, where $\bar{x} = \frac{1}{|C|} \sum_{x_i \in S_C} x_i$ and $s = \sqrt{\frac{1}{|C|-1} \sum_{x_i \in S_C} (x_i - \bar{x})^2}$.

Definition 1. (*Homogeneous cluster*) A cluster C is said to be *homogeneous*, if the skewness and kurtosis of random variable X_C satisfy the following criterion:

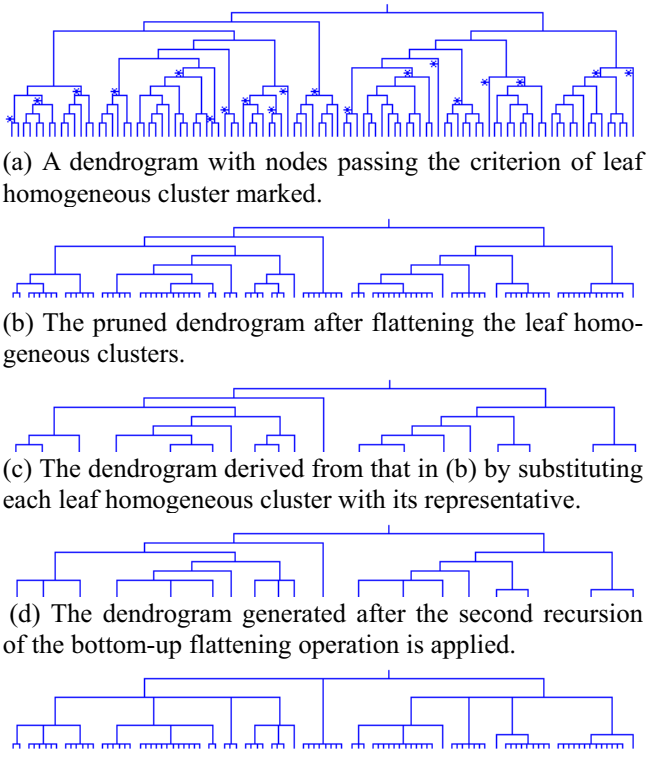
$$-\theta_s \le Skew(X_C) \le \theta_s \text{ , and}$$
$$Kurt(X_C) \ge \theta_k ,$$

where θ_s and θ_k are two thresholds to be set by the user. In this paper, θ_s and θ_k are set to 1 and 0, respectively.

Though θ_s and θ_k are global thresholds, they are imposed on statistical measures that can catch local variation of the pairwise similarity scores among data instances. In order to calculate $Skew(X_C)$ and $Kurt(X_C)$ efficiently, we maintain some statistics in each node, X_C, of the dendrogram. They are the first four moments of the distribution of pair-wise similarity: E[X_C], E[X_C ^{2}], E[X_C ^{3}] , and E[X_C ^{4}], where $x_i \in S_C$ stands for the pair-wise similarity of points in a cluster. The definition for moment can be found in [8].

Definition 2. (*Leaf homogeneous cluster*) A cluster C in a dendrogram is said to be a *leaf homogeneous cluster*, if C and all of its children are homogeneous clusters and the parent of C is not satisfied as a leaf homogeneous cluster.

In the summarization process, one object in a homogeneous cluster will be designated as the representative of the cluster. The representative of a homogeneous cluster C is the object with the maximum lumped sum of the similarity scores to the other objects in C. In this paper, the representative of cluster C is denoted by $Rep(C)$.



(a) A dendrogram with nodes passing the criterion of leaf homogeneous cluster marked.

(b) The pruned dendrogram after flattening the leaf homogeneous clusters.

(c) The dendrogram derived from that in (b) by substituting each leaf homogeneous cluster with its representative.

(d) The dendrogram generated after the second recursion of the bottom-up flattening operation is applied.

(e) The final dendrogram.

Fig. 2. An example illustrating the flattening operation.

With homogeneous clusters and their representatives identified, the summarization process then conducts a bottom-up flattening operation on the dendrogram generated by the agglomerative hierarchical clustering algorithm. The bottom-up flattening operation begins with the leaf homogeneous clusters. All the subclusters under a leaf homogeneous cluster will be removed and all the objects contained in the leaf homogeneous cluster will become its children. Fig. 2 illustrates the bottom-up flattening operation. Fig. 2(a) shows the initial dendrogram constructed with 100 objects. In Fig. 2(a), those clusters marked by an asterisk (∗) are leaf homogenous clusters according to definition 2. Fig. 2(b) shows the dendrogram after the bottom-up flatten-

ing operation has been applied to the leaf homogeneous clusters.

The bottom-up flattening operation described above is conducted recursively with each of the leaf homogeneous clusters identified in one level of recursion being substituted by its representative object. Fig. 2(c) depicts the dendrogram derived from substituting the leaf homogeneous clusters in Fig. 2(b) with their respective representatives and Fig. 2(d) shows the flattened dendrogram after the second level of recursion is conducted. Fig. 2(e) shows the final dendrogram after the bottom-up flattening operation is completed and this dendrogram is referred to as the abstract dendrogram.

3.2 The incremental process

With the flattened dendrogram, the incremental phase of the clustering algorithm is then carried out to cluster the remaining objects in the database, i.e. those objects that are not taken to construct the initial dendrogram, as well as the objects that may be added into the database later on. These objects are examined one by one. For each object, the incremental clustering algorithm examines whether the object can be inserted into a leaf homogeneous cluster according to the following criteria. The criterion for inserting an object p into a leaf homogeneous cluster C is as follow:

$$\left|\frac{\mu_C - sim(p, Rep(C))}{\sigma_C}\right| \le \theta_q \text{ and}$$

$$sim(p, Rep(C)) \ge \forall_{C'} sim(p, Rep(C')), \text{ where}$$

(1) μ_C and σ_C are the mean and standard deviation of the pairwise similarities in C;
(2) C' is a leaf homogeneous cluster of size larger than 1,
(3) $sim(p, Rep(C))$ denotes the similarity between object p and object $Rep(C)$,
(4) θ_q is a parameter and is set to 1 in this paper for carrying out clustering.

Each time when an object is inserted into a leaf homogeneous cluster, the skewness and kurtosis of the cluster will be recomputed to check whether the cluster still meets the criterion of being homogeneous. Should the cluster, with the object inserted, fails to pass the test, a *split* operation will be conducted. A hierarchical agglomerative clustering (HAC) algorithm and the flattening operation described in section 3.1 will be invoked to construct a sub-dendrogram containing all the objects in the cluster and the newly added object. The split operation and the reconstruction operation described in next paragraph are essential for avoiding order dependence, which is a common problem in many incremental clustering algorithms.

In case the object being examined cannot be inserted into any of the existing leaf homogeneous clusters, then the object is temporarily moved to a temporary buffer called *TempBuffer* and will be processed again later on. Each time when *TempBuffer* becomes full, a reconstruction operation is conducted to generate a new abstract dendrogram containing all the objects in the current dendrogram and the objects in *TempBuffer*. In the reconstruction operation, the primitive objects are the representatives of the leaf homogeneous clusters and the objects in *TempBuffer*. In this paper, any hierarchical agglomerative clustering algorithm can be invoked to the reconstruction process. During the reconstruction process, two leaf homogenous clusters in the original dendrogram may merge due to inclusion of the objects in *TempBuffer*. As mentioned earlier, this effect along with the split operation described above are essential for avoiding order dependence.

3.3 Time complexity

The completed analysis of time and space complexities of the proposed incremental clustering algorithm can be found in [6]. In summary, the time complexity is $O(km)+O(kq^2\log q)+O((k/b\_s)m^2\log m))$, where

(1) n: the number of the proteins in the current version of the protein database;
(2) k: the number of new proteins to be added to the protein database;
(3) m: the number of leaf homogeneous clusters in the current version of the abstract dendrogram;
(4) q: the number of proteins that the largest leaf homogeneous cluster contains;
(5) b_s: the size of the *TempBuffer*.

As the experiments reported in [6] reveal, for cluster analysis of contemporary protein databases, we generally have $q<<m$. Therefore, the dominant term of the time complexity for carrying out protein sequence clustering with the proposed incremental clustering algorithm is $O(km)+O((k/b\_s)m^2\log m))$ or $O(km^2\log m)$, if b_s is regarded as a constant.

4. Experiments

In this section, we report the result of using alignment scores as similarity measure and employing single-link algorithm as the HAC invoked by the incremental clustering algorithm to demonstrate the effectiveness and efficiency of the proposed incremental algorithm. The reason why this combination, sequence alignment accompanied with single-link algorithm, is shown here is due to its widely used in protein sequence clustering for function analysis. In this paper, the quality of the clustering result is evaluated based on how well the clustering algorithm is able to cluster proteins in conformity with the protein families defined in InterPro [2]. The quantitative measure used is the weighted average matching rate defined by the following equation:

$$\overline{M}(D) = \frac{1}{\sum_{F_i} |F_i \cap S|} \sum_{F_i} |F_i \cap S| \cdot M(F_i, D),$$

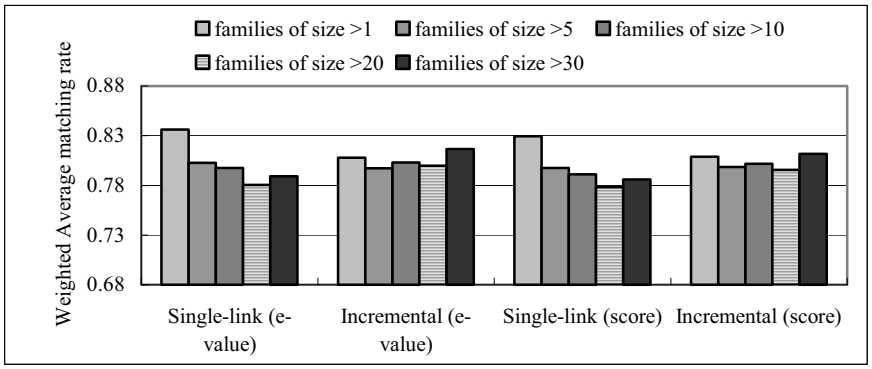


Fig. 3. Lumped averages of the experimental results on the three datasets.

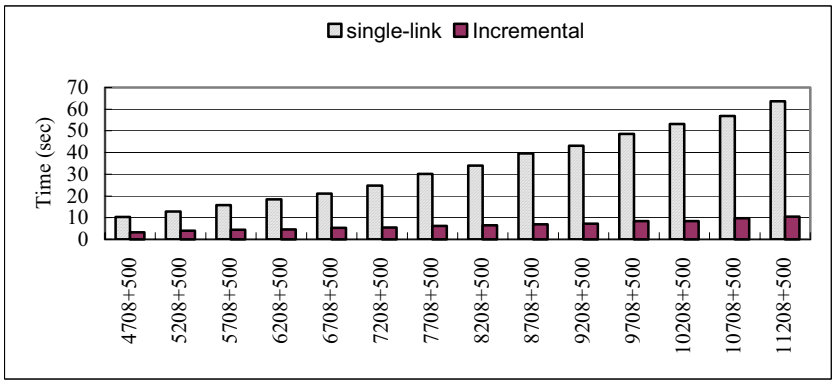


Fig. 4. Comparison of the execution times of the proposed incremental clustering algorithm and the single-link algorithm.

In the above equation, the matching rate of a particular family F defined in the InterPro with respect to a dendrogram D generated by a clustering algorithm is defined as:

$$M(F,D) = \max_{C_i \in D} \frac{|C_i \cap (F \cap S)|}{|C_i \cup (F \cap S)|},$$

where C_i is a cluster in dendrogram D, i.e. a node in the dendrogram, S is the set of proteins on which clustering is conducted, and |A| stands for the number of objects in set A.

Table 1 summarizes the characteristics of the three datasets used in this study. The three datasets contain the proteins belonging to mouse, rat, and human in Swiss-Prot [4] (Release 40, October 2001), respectively. Fig. 3 depicts the bar charts of the overall weighted average matching rates that the proposed incremental clustering algorithm and the single-link algorithm deliver in clustering these three datasets. In this experiment, both the bit-score and e-value computed by the BLAST algorithm [1] with BLOSUM62 table are employed. As the E-values between two protein sequences are asymmetric and the dynamic range of the E-values is extremely large, the following transformation is applied to compute the similarity scores based on the E-values:

$sim(a, b)$ = – ln(geometric average of the E-values between proteins a and b),

where $sim(a, b)$ stands for the similarity score between protein a and protein b. The numbers corresponding to the proposed incremental clustering algorithm are the averages of 5 independent runs with random input sequence. As the results presented in Fig. 3 shows, the proposed incremental clustering algorithm performs more consistently than the single-link algorithm in dealing with proteins belonging to families of various sizes.

Fig. 4 compares the actual execution times of the proposed incremental clustering algorithm and the single-link algorithm in one benchmark case. In this experiment, 7000 human proteins in Swiss-Prot are incrementally added into a dataset that contains all the 4708 mouse proteins in Swiss-Prot. Each time 500 proteins are added and the single-link algorithm needs to generate a new dendrogram from scratch. On the other hand, the proposed algorithm resorts to an abstraction generated by the previous run of the algorithm to carry out clustering incrementally. As Fig. 4 reveals, due to the incremental nature, the proposed incremental clustering algorithm takes less than 20% of the time taken by the single-link algorithm to generate a new dendrogram for cases in which n > 10,000. The execution time reported in Fig. 4 does not include the time taken to compute the pairwise similarity scores among proteins.

Table 1. Summary of the three datasets used in the experiments.

Dataset	Number of proteins	Number of cross-referenced InterPro Families	Number of proteins labeled with family identification
Mouse	4708	861	2563
Human	7471	1067	3796
Rat	2916	714	1902

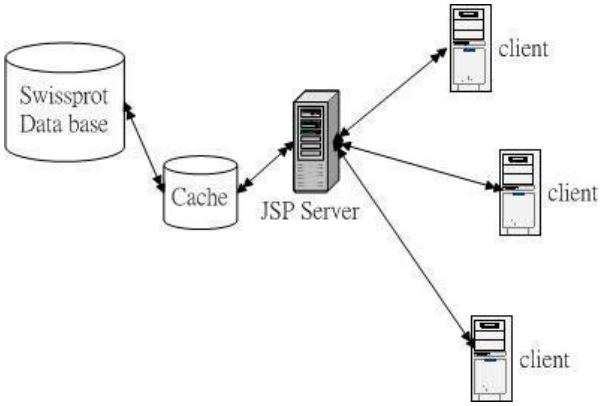


Fig. 5. The system architecture for the clustering utility.

5. System architecture and implementation

5.1 System architecture

Fig. 5 shows the system architecture. The utility is implemented with java solutions. First of all, Tomcat is employed as the servlet container. Tomcat is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies. In order to have the services with high performance, we use caching mechanism to store the results for often asked queries.

We can see the system flow in detail in Fig. 6. The operations executed in each step will be examined clearly as follows.

1. A user sends a request, which includes the interested motif, the interested species in the taxonomy, the type of similarity measure, the employed hierarchical clustering algorithm, and email information (optional).
2. Server scans the local database for proteins with the specified motif.
3. The similarity scores are calculated or looked up according to the request.
4. The server prepares data for further clustering process.
5. The server receives the result from the clustering process, and caches the result.
6. The server triggers the client to create applet for viewing the tree.
7. The client requests the dendrogram from the server.
8. The client receives data and displays it.

Note that, the system will both cache the result of searching the database for proteins with the specified motif at the end of step 2, and cache the outcome of clustering process in step 4. In other words, before step 2, the server will check whether the current query has been requested before. If so, the system will go for a cached file for clustering result. If the file exists and is valid, the server will skip the following steps, step 2, step 3, step 4, and step 5, and pop the result back to the client directly. If there were only the cached file for the result of searching database, the system would just skip step 2 and step 3, and continue to process step 4 and step 5.

If the user intends to view the clustering results according to annotations, the server will get further information from the local database to generate feature vector for each protein. First, as shown in Fig. 7, we split CC field into topic portion and description portion. After that, we ignore the topic part, and remove the stop words and the words that appear only once from description part. Finally, as mentioned in section 2.1, the inverse-document-frequency (IDF) of a term is used for weighting the features.

On the other hand, if the user is interested in viewing the results from the aspect of sequence similarity, we use the alignment scores calculated by BLAST program [1] to cluster the selected proteins. The all-against-all similarities are calculated in advance, but only similarity scores with E-value smaller than 100 are stored by the system.

5.2 Implementation issue

The system is going to serve many requests from several clients, so the performance is a great important issue. Performance is normally related to CPU and memory. Thus, how to save CPU time and memory is the key issue when developing the clustering utility. If we have kept the result of a previous request, we do not compute again for the same request. In other words, CPU power can be saved. This skill is called "caching", which is already a famous solution for computer architecture and Internet technology.

Besides, each clustering process requires a large size of memory if all the similarity scores are loaded when performance is considered. If we make every process allocate large size of memory, the system memory will be exhausted quickly. In order to prevent this situation, we employ the shared memory mechanism, letting all the clustering processes read the same location of memory for similarity scores. In the following subsections, we will explain these two mechanisms in detail.

5.2.1 Shared memory

Shared Memory is an efficient way to pass data or information between programs. One particular program will create a memory portion that other processes (if permitted) can access. Shared memory is the fastest form of IPC available. Once the memory is mapped into the address space of the processes that are sharing the memory region, no kernel involvement occurs in passing data between the processes. What is normally required, however, is some form of synchronization between the processes that are storing and fetching information to and from the shared memory region.

What we mean by "no kernel involvement" is that the processes do not execute any system call into the kernel to pass the data. Obviously, the kernel must establish the memory mappings that allow the processes to share the memory, and then manage this memory over time (handle page faults, and the like).

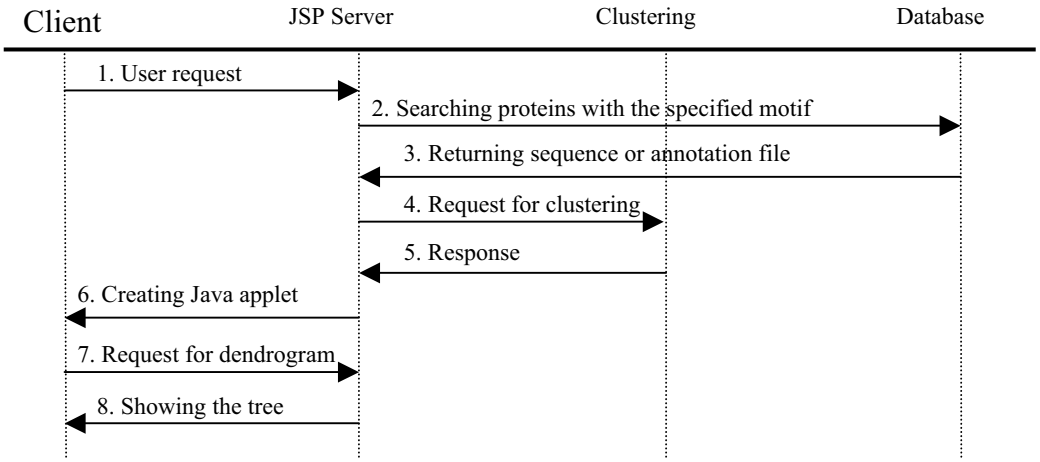


Fig. 6. The overall view of the system flow.

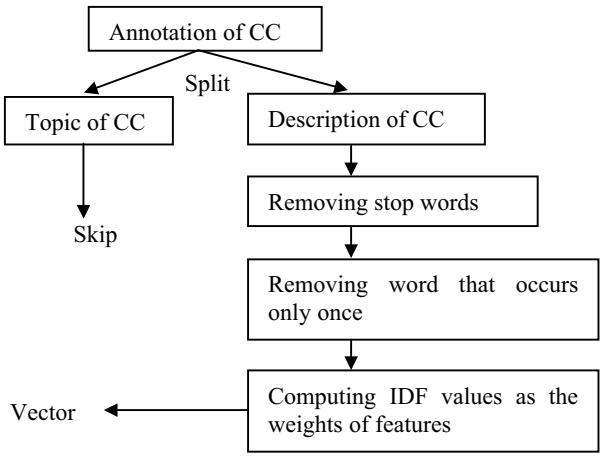


Fig. 7. The procedure of step 4 in the case when the functional annotation is selected as the feature.

5.2.2 Caching architecture

The CPU time can be greatly saved by using cache mechanism. However, like in many process environments, it has a problem of race condition when using cache. When several processes access and manipulate the same data concurrently, the outcome of the execution depends on the particular order in which the accesses take place. This is called a race condition.

We settle this problem by the following rule. Fig. 8 depicts the overview of our solution. Since the procedure of checking the cached files serves several processes, it will encounter race condition problem. The collection of techniques for sharing resource such that different uses do not conflict and cause unwanted interactions is referred to as Mutual Exclusion techniques. Action *a* of process A and action *b* of process B must exclude each other if the execution of process A may not overlap the execution of process B. When processes A and B execute in parallel, one of the processes must be blocked until the other completes its action. A sequence of actions that must be protected from interleaving is called critical section. So this procedure must be in critical section.

The situation when checking the cached files comes in three cases. If the desired file has not been cached, the procedure of clustering and caching results for this request are processed. If the desired file exists, but this file hasn't been completed, clustering procedure is processed and the result will be delivered to the client. If the desired file exists and has been finished, the server just delivers the result to the client.

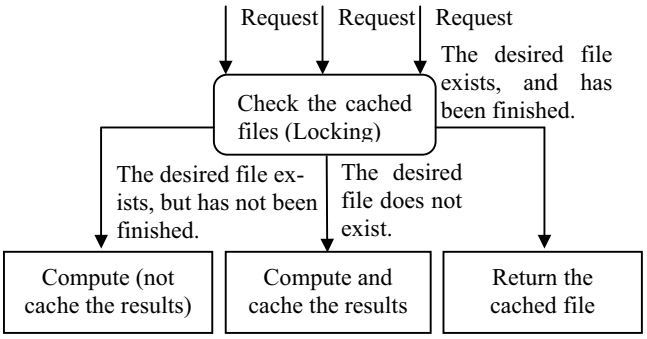
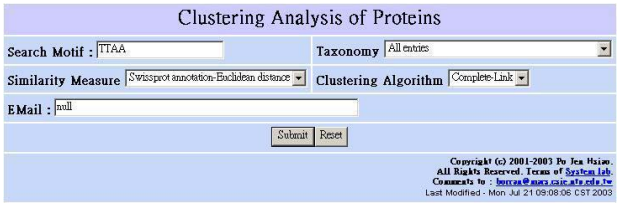
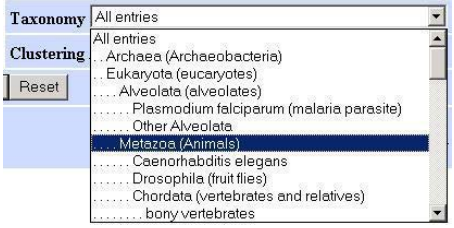


Fig. 8. Caching architecture



(a) The query page.



(b) Option for selecting a particular species.

Fig. 9. The web page of the protein clustering utility.

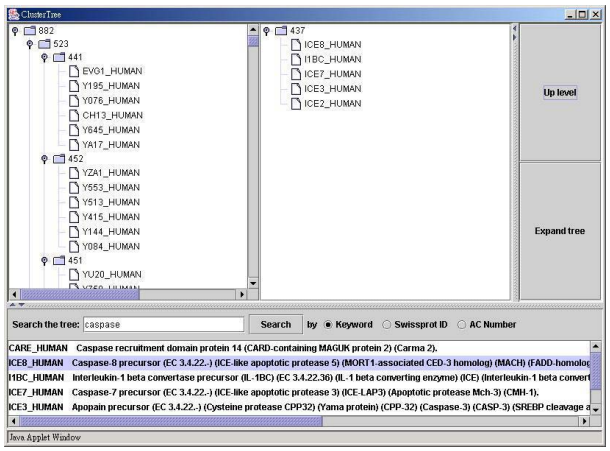


Fig. 10. Tree visualization

5.3 Usage and visualization

The protein clustering utility described in this paper is currently available at:
http://uranus.csie.ntu.edu.tw:9000/index.jsp . As shown in Fig. 9(a), the user can specify the species of interest and select the desired similarity measure and clustering algorithm. Fig. 9(b) shows a portion of the taxonomy employed in the utility. This taxonomy is derived from the OC (organism classification) field in Swiss-Prot [4].

Fig. 10 shows the visualization for the dendrogram generated by each clustering process. Each leaf node is presented with the protein name that is given by Swiss-Prot database. By the query box in the bottom of the window, the user can search proteins by keywords, the Swiss-Prot entry-name, or the primary AC number. After that, the system will list all the proteins satisfying the query, and will show the result in the right box after the user specifies a particular protein in the list. If the user wants to see the parent node of the selected protein, he/she can clink the "UP LEVEL" panel to trace up.

6. Conclusion

Motif databases provide powerful tools for analyzing uncharacterized sequence data, in particular by placing individual sequences in a family context for a more informed assessment of function than is possible with conventional pairwise searches. While there is overlap between them, the contents of the family databases differ. It is therefore good to use InterPro because it is an integrated documentation resource for protein families, domains and functional sites. In our developed system, we compared with the results from InterPro database, and got the rate that showed the query result.

Using database search and sequence clustering methods, we have developed this clustering utility which can identify and categorize many protein families within various species such as *Homo sapiens*, *Drosophila melanogaster*, *Caenorhabditis elegans*, *Saccharomyces cerevisiae*, and so on. We can choose which species is favored in Taxonomy; therefore, we can narrow down the search range.

This clustering utility addresses and fulfils a need in the bioinformatics community namely doing analysis on the selected set of entries based on motif sequence. It also is very useful for motif hunters and database curators. The employed incremental clustering algorithm is able to identify protein sequence clusters that match protein families more consistently than the single-link algorithm while execution times is largely reduced. In the future, we will in further make use of the advantage of incremental clustering algorithm that is able to cluster new sequences from the user with the existing dendrogram containing all the proteins in a large database, like Swiss-Prot.

7. Acknowledgements

This work was supported by a grant from National Science Council.

References:

[1] Altschul, S.F., Madden, T.L., Schaffer, A.A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D.J. (1997) Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Res. 25, 3389-3402.

[2] Apweiler, R., Attwood, T.K., Bairoch, A., Bateman, A., Birney, E., Biswas, M., Bucher, P., Cerutti, L., Corpet, F., Croning, M.D.R., Durbin, R., Falquet, L., Fleischmann, W., Gouzy, J., Hermjakob, H., Hulo, N., Jonassen, I., Kahn, D., Kanapin, A., Karavidopoulou, Y., Lopez, R., Marx, B., Mulder, N.J., Oinn, T.M., Pagni, M., Servant, F., Sigrist, C.J.A., Zdobnov, E.M. (2000), InterPro - an integrated documentation resource for protein families, domains and functional sites, *Bioinformatics*, **16**, 1145-1150.

[3] Baeza-Yates, R. and Ribeiro-Neto, B. (1999) *Modern information retrieval*, Addison-Wesley.

[4] Bairoch, A. and Apweiler, R. (2000) The Swiss-Prot protein sequence database and its supplement TrEmBL in 2000, *Nucleic Acids Res.*, **28**, 45-48.

[5] Blevins R., Aaronson J., Myerson J., Hamm G., and Elliston K. (1995) PROFILER: a tool for automatic searching of internally maintained databases, *Comput Appl Biosci*, **11**, 667-673.

[6] Chen, C.-Y., Oyang, Y.-J., and Juan, H.-F. (2003) An Incremental Clustering Algorithm for Protein Database Analysis, *Technical Report of CSIE-NTU*, 03-01.
(http://mars.csie.ntu.edu.tw/~cychen/NTUCSIE-03-01.pdf)

[7] Jain, A.K., Murty, M.N., Flynn, P.J. (1999) Data Clustering: A Review, *ACM Computing Surveys*, **31**, 264-323.

[8] Jobson, J.D. (1991) *Applied Multivariate Data Analysis*, Springer-Verlag.

[9] E.V. Kriventseva, W. Fleischmann, E.M. Zdobnov, R. Apweiler (2001) CluSTr: a database of clusters of Swiss-Prot+TrEMBL proteins, Nucleic Acids Res., 29, 33-36.

[10] McLachlan A. D. (1980) Repeated folding pattern in copper-zinc superoxide dismutase, *Nature*, **285**, 267-268.

[11] Puntervoll, P., Gemund C., and *et al* (2003) EML server: a new resource for investigating short functional sites in modular eukaryotic proteins, *Nucleic Acids Research*, **31**,

lar eukaryotic proteins, *Nucleic Acids Research*, **31**, 3625- 3630.